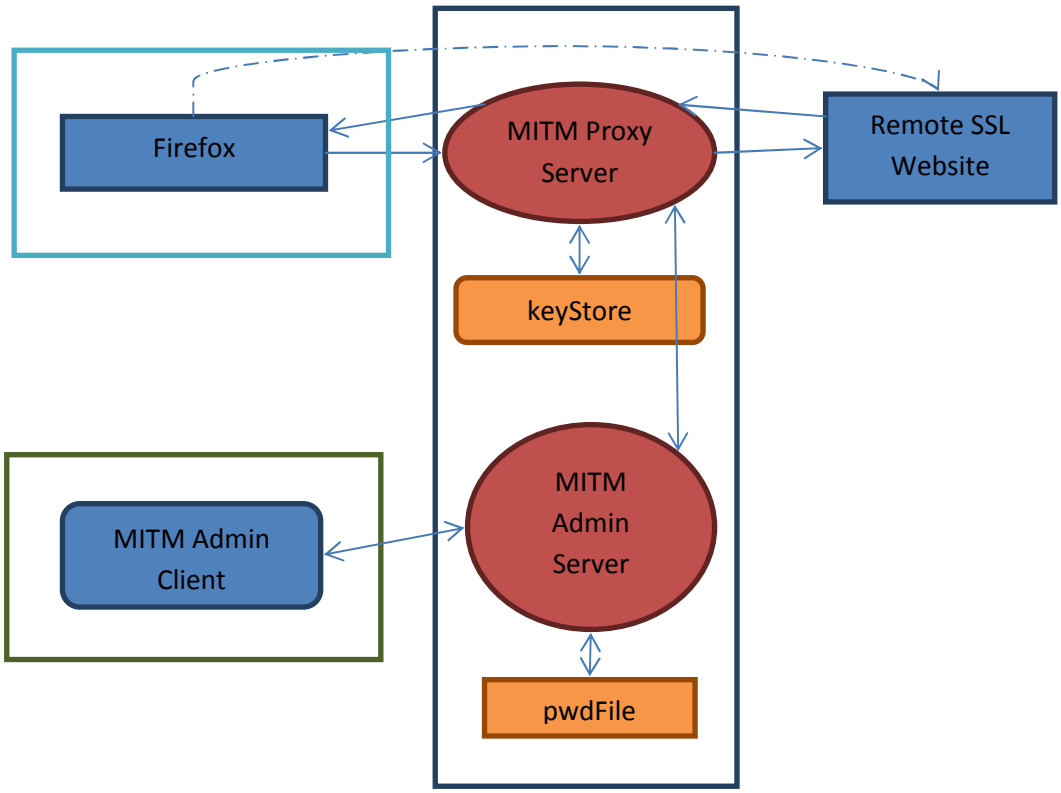
Project 2 – SSL MiTM

Hai Xiao (SUNetID: haixiao) and Amit Chattopadhyay (SUNetID: amitch)

Threat Model Design Diagram:



Design Decisions:

We could do either MAC or RSA based challenger-response admin client authentication if had more time, as an alternative we could also have done SSL/TLS client authentication during handshake between AdminServer←→AdminClient!

- **KeyStore**

  The keyStore of our proxy is generated using the Java keytool with the following parameters:

  Key Gen Algorithm: RSA
  Key Length: 1024 bit
  Signing Algorithm: MD5withRSA (default), also use SHA1withRSA
  Alias: mykey

  Keypass/storepass: cs255key

  The Java keystore here is used as a repository of RSA certificates, public keys and private keys. It is vitally important and confidential. The entire keystore is protected with integrity by storepass. In addition the private keys in repository are protected with keypass.

Project 2 – SSL MiTM

Hai Xiao (SUNetID: haixiao) and Amit Chattopadhyay (SUNetID: amitch)

- **MITM Proxy Server**

  The core functionality is present in MITMSSLSocketFactory() where we intercept the serverDN and serialNumber. The proxy server will create a forged server certificate for the MiTMed SSL/TLS sessions, which looks like proxy server's own original certificate but with the real remote server's serverDN and serialNumber (and can add other fields, but not due to time constraints), then it is resigned with the private key of the MITM Proxy Server. The certificate and private key of the Proxy Server is retrieved from the its keystore which is protected using a keystore password.
  This new dynamically generated server certificate chain is then used at client/proxy-server handshake.

- **Admin Server/Client**

  The MITM Admin Server accepts incoming connections from a MITM Admin Client that is authenticated via a password that is sent through SSL/TLS channel. We are using bcrypt library to perform password hashing and checking. Bcrypt is a reputable cryptographic algorithm which performs a slow hashing preventing a timing attack on the password by an attacker and bot login attempts. The bcrypt library provides an implementation that takes a salt and a string and generates a salted hashed. By default the gensalt() method performs 10 rounds which is considered secure and we are using the default setting. (We could use higher numbers of iteration) This salted hash is then written to pwdFile.
  (reference: http://security.stackexchange.com/questions/4781/do-any-security-experts-recommend-bcrypt-for-password-storage)
  The MITM Admin Client must communicate the Password and Command to the MITM Admin Server. This communication takes place through a Secure Connection which we establish using a separate key from the one that is used to communicate with incoming client connections to the Proxy Server.

Deployment/Use Steps:

Proxy Server

1. Open passwords.txt and find the keystore password.
2. Build
   a. ./setup.bash
   b. make clean; make
3. Run the MITMProxyServer
   a. **java mitm.MITMProxyServer –keyStore keystore –keyStorePassword cs255key –outputFile logfile**
   b. Ensure the proxy server started up.

Proxy Client:

1. Start Firefox
2. Setup SSL Proxy as appropriate
3. Connect to HTTPS website.

Project 2 – SSL MiTM

Hai Xiao (SUNetID: haixiao) and Amit Chattopadhyay (SUNetID: amitch)

Admin Client:

1. Open passwords.txt and find the admin server password.
2. Build
    a. ./setup.bash
    b. make clean; make
3. Run the AdminClient
    a. Java mitm.MITMAdminClient –password cs255project -cmd [stats | shutdown]
    b. Ensure correct output.

Security Aspects:

1. Never had keystore password stored anywhere (except for passwords.txt for grading) in system. This provides a robust protection to keystore so to key materials to the entire system in design.
2. In practice, we should have all our key material or security parameters stored only in one keystore file, guarded (with encryption and integrity) by Java JCA framework. In this project we have not done symmetric encryption or MAC, etc. explicitly in our code, if we do, we will store the keys in the same keystore.
3. We provided many-time password authentication for admin client, but we strength it with running login over SSL/TLS sessions with salted password hash verification done by BCrypt.
4. Ideally we should provide password in passwords.txt not in language word forms, but in high entropy only to prevent dictionary or rainbow attacks. We didn't do this even we should; we understand that there is no good reason not to do this, probably due to time spent elsewhere.
5. To MiTM hijacked SSL/TLS sessions, we never stored the dynamical allocated serverKeyStore anywhere on system, except in memory! It is important as otherwise we will most likely leak original private key, etc. information (if disk/file is decrypted) later on, and prone to forward security attacks! We also gain performance benefits by not saving to storage.
6. We also provide a protection password to each dynamically allocated serverKeyStore so private key materials are secure. We made this password the same as keypass to MiTM proxy's keystore to keep simple and relay a robust security to a same root security parameter.

Project 2 – SSL MiTM

Hai Xiao (SUNetID: haixiao) and Amit Chattopadhyay (SUNetID: amitch)

Short Answers to Questions:

1. Suppose an attacker controls the network hardware and can intercept or redirect messages. Show how such an attacker can control the admin server just as well as a legitimate admin client elsewhere on the network. Give a complete and specific description of the changes you would make to fix this vulnerability.

Answer: This is called MiTM (Man in the Middle) attack. Specific to this case it is SSL/TLS MiTM attack between admin client and admin server SSL/TLS communication channels. The method an attacker uses to run this attack is exactly the same with that we implemented in this project (if attacker can cheat a legitimate admin client to 'add an exception' for the forged certificate to user end browser) but being applied to ourselves! There are other ways that attacker can incorporate to achieve the similar goal, such as break into the SSL/TLS session and steal security tokens (such as login password) using CBC padding attack or take advantage of RC4 cipher vulnerability. But we are trying to focus our discussion based on MiTM attack here (Assume we had strength our server not use CBC mode or RC4 cipher). Start with MiTM attack, if it is successful, then attacker can see all cleartext (may be compressed, but even through still decomp-able) message exchange between admin client and admin server, including the login password, so later on attacker can even login admin server just like admin user with given admin's privilege, damage can be unpredictable based on what that privilege is!

Our implementation so far just focused on how to achieve this SSL/TLS MiTM attack between client and server for traffic running through our controlled SSL/TLS proxy server, but so far we have not avoid MiTM attack over our proxy server admin/management interface (admin client ←→admin server), since we are using the password authentication only, we should know that this (even stronger one-time password) along could not prevent MiTM attack!

To fix this vulnerability, with good reason we need to implement challenger-response authentication to provide stronger bidirectional authentication, so MiTM won't be able to commence attack and decrypt traffic. Details and options are: (1) Specific to SSL/TLS, we can make proxy server (admin server end) always to authenticate admin client by request SSL/TLS 'client auth' during handshake, instead of one-way authentication (server auth to client) previously. (During Lecture Dan had mentioned that there are some challenges for clients to manage their private keys efficiently, but it can be done with specific methodologies to specific situations, such as kept in personal key-dongle if device share is concern, etc.) (2) There are other none-generic ways to fix this, such as using public-key pinning to browsers (supposedly admin user's browser), but we cannot make assumption that admin client users only use a certain browser from a certain device, so this solution is not generic.

Fix detail: (1) Need to add SSL/TLS client authenticate code to admin secure socket interface. (2) Or we can add an out-bind challenger-response protocol given admin server knows admin client's public key.

Project 2 – SSL MiTM

Hai Xiao (SUNetID: haixiao) and Amit Chattopadhyay (SUNetID: amitch)

2. Suppose an attacker is trying to gain unauthorized access to your MITM server by making its own queries to the admin interface. Consider the security of your implementation against an attacker who (a) can read the admin server's password file, but cannot write to it; (b) can read and/or write to the password file between invocations of the admin server. For each threat model, either show that your implementation is secure, or give an attack. (N.B.: For full credit, your implementation should at least be secure under (a).) What, if anything, would you need to change in order to make it secure under (b)? If your answer requires any additional cryptographic tools, you should fully specify them (including the names of any algorithms, cryptosystems, and/or modes of operation that you would use.)

Answer:

(a) Threat model:

Attacker can read **pwdFile** content, and attacker is trying to gain admin server login with good password guess given **pwdFile** is not a secret!

(a) Our implementation:

Is secure! Reason being:

1. File **pwdFile** does not contain any cleartext password or salt information.
2. File **pwdFile** only contains a salted password generated from BCrypt library, supposedly secure.
3. The BCrypt implementation has done many hash iterations to make bad login attempts slower.
4. The random salt chosen by BCrypt should have very good entropy to back up security.
5. The login password is never stored in clear anywhere on proxy server and other systems.
6. Also <password> is only provided over SSL/TLS secure session, so excluding MiTM attacks from question 1. attacker has no way to figure out login <password> by eavesdropping (exclude CBC padding or RC4 attacks)
7. So even attacker sees **pwdFile** content, there is no other way to attack other than brute-force (or dictionary) search, given BCrypt salted and many-time iteration hash, the attack is inefficient.

(b) Threat model:

Attacker can read and write **pwdFile** content, so attacker is trying to gain admin server login with good password guess or use a password he/she designs (attacker can choose a password, then run the same BCrypt code <can be offline> to generate salted hash of this forged password, then swap the content of original **pwdFile** with this forged content), provided that attacker can also manipulate **pwdFile**!

(b) Our solution (not implemented given limited time):

To make design robust to Threat (b) we have below options (Disclaimer: we are not to avoid DoS attack):

1. Add signature to **pwdFile** signed with admin server's private key (e.g. signing with algorithm sha1RSA. In fact not only admin server can sign file **pwdFile**, any other trusted entity can sign also so long as admin server may obtain corresponding public key during verify). In this case if **pwdFile** content or signature be tampered, admin server finds out right away so to fail login!
2. An inferior alternative method would be to use MAC or AE to guard **pwdFile** 's integrity. But it incurs extra keys be managed by admin or proxy server, so makes it less clean than option 1.
3. Required any cryptographic tools, algorithms, cryptosystems, modes of operations (you should fully specify them): **Java JCA/JSSE/etc., keytool, None CBC mode block ciphers, None RC4 ciphers, SSL/TLS client authentication, RSA key exchange or even EDH key exchange, etc.**

Project 2 – SSL MiTM

Hai Xiao (SUNetID: haixiao) and Amit Chattopadhyay (SUNetID: amitch)

3. How would you change a web browser to make it less likely that an end user would be fooled by a MITM attack like the one you have implemented? (This is an important question to ask because when dealing with security, we never just build attacks: we also need to think of ways to prevent them.)

Answer: Don talked in lecture, there are couple of method to harden end user browser to make MiTM attack less likely to happen, those include cert/public-key pinning as in Chrome, HTTP header pub-key pinning, OCSP reinforce and fallback change, add client certificate auth with hardware keystorage, etc.

But to our simple MiTM implementation, there are other easier ways if to modify a end user's web browser to avoid MiTM attacks, for example:

1. Remove any 'Add security exception' hook at all.
2. Do more checks over various certificate fields, e.g. check Issuer's Identification.
3. Do more stateful checks, e.g. check for duplicated public keys from multiple certs or domains.
4. Add hooks (if not yet) to statically add a specific certificate/public-key to site/domain mapping.
5. Etc.